

Exhibit 5

U.S. Patent No. 8,190,610

Booking.com B.V.



Claim Chart for Representative Claim 1

References Cited

Upon information and belief, Booking.com has and continues to use a system for big data analytics (hereinafter, the “Booking.com System”), which is a system based on Apache Hadoop with Hive and Spark that includes the elements in the systems described in and/or practices the steps of the methods described in the claims of U.S. Patent No. 8,190,610 (the “610 Patent”).

The following chart presents R2 Solutions’ analysis of the Booking.com System based on publicly available information. The citations in the chart refer to the following publicly available documents, which are incorporated by reference as if fully set forth herein:

- [1] Sachin P. Bappalige, “An introduction to Apache Hadoop for big data,” opensource.com (Aug. 26, 2014), *available at* <https://opensource.com/life/14/8/intro-apache-hadoop-big-data> (“**An introduction to Apache Hadoop for big data**”)
- [2] MapReduce Tutorial, *available at* <http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html> (“**MapReduce Tutorial**”)
- [3] Interface Reducer<K2,V2,K3,V3>, *available at* <https://hadoop.apache.org/docs/r2.8.0/api/org/apache/hadoop/mapred/Reducer.html> (“**Interface Reducer**”)
- [4] ASF Infrabot, “HadoopMapReduce,” Dashboard (July 9, 2019), *available at* <https://cwiki.apache.org/confluence/display/HADOOP2/HadoopMapReduce> (“**HadoopMapReduce**”)
- [5] Booking.com Lead Data Engineer, *available at* <https://able.bio/bookingcom/jobs/lead-data-engineer-enterprise-data-management-remote/4bb2f3> (“**Booking.com Lead Data Engineer**”)
- [6] “Tushar Kesarwani,” Booking.com employee LinkedIn page, *available at* <https://www.linkedin.com/in/tushar-kesarwani/?originalSubdomain=nl> (“**Booking.com Data Engineer**”)

References Cited (continued)

- [7] Hive 2.1.1 API – Class Schema, *available at* <https://hive.apache.org/javadocs/r2.1.1/api/org/apache/hadoop/hive/metastore/api/Schema.html> (“**Class Schema**”)
- [8] Confluence Administrator, “Hive Tutorial,” Dashboard (updated by Owen O’Malley, Mar. 5, 2019), *available at* <https://cwiki.apache.org/confluence/display/Hive/Tutorial> (“**Hive Tutorial**”)
- [9] Confluence Administrator, “LanguageManual Joins,” Dashboard (updated by Dudu Markovitz, June 5, 2017), *available at* <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins> (“**LanguageManual Joins**”)
- [10] Tanel Poder, “Speed Up Your Queries with Hive LLAP Engine on Hadoop or in the Cloud,” Gluent, *available at* <https://www.slideshare.net/gluent/speed-up-your-queries-with-hive-llap-engine-on-hadoop-or-in-the-cloud> (“**Speed Up Your Queries with Hive LLAP Engine**”)
- [11] “Apache Hive,” Wikipedia, *available at* https://en.wikipedia.org/wiki/Apache_Hive (“**Apache Hive**”)
- [12] Petar Zecevic & Marko Bonaci, “Spark in Action” (Manning Publications Nov. 2016), *available at* https://learning.oreilly.com/library/view/spark-in-action/9781617292606/kindle_split_014.html (“**Spark in Action**”)
- [13] “Apache Spark FAQ,” *available at* <https://spark.apache.org/faq.html> (“**Apache Spark FAQ**”)
- [14] “Daniel Serrano,” previous Booking.com Employee LinkedIn Profile, *available at* <https://www.linkedin.com/in/daniel-serrano-gine/?originalSubdomain=uk> (“**Booking.com Big Data Engineer**”)
- [15] Manager Software Development, *available at* <https://startup.jobs/manager-software-development-payments-bookingcom-671446> (“**Booking.com Manager Software Development**”)
- [16] “Booking.com Privacy Statement,” *available at* <https://www.booking.com/content/privacy.html?label=gen173nr-1FCAEoggl46AdIM1gEaLACiAEBmAExuAEXyAEM2AEB6AEB-AECiAlBqAIDuAK1w--MBsACAdICJGQyYTA4YmYOLWRkZjktNDJmNy04MzgZLTUONTlwZmY0NWY1Y9gCBeACAAQ;sid=b71ed236bdaa2f99b0b2eec2c15415df;sig=v1m-R6d64h#personal-data-collected-type> (“**Booking.com Privacy Statement**”)

References Cited (continued)

- [17] Apache Spark 3.0.1, “RDD Programming Guide,” *available at* <http://spark.apache.org/docs/latest/rdd-programming-guide.html> (“**RDD Programming Guide**”)
- [18] “Learning Spark,” O’Reilly Media, Inc. (February 2015), *available at* <https://learning.oreilly.com/library/view/learning-spark/9781449359034/ch04.html#chap-pair-RDDS> (“**Learning Spark**”)
- [19] “Spark RDD Transformations with examples,” *available at* <https://sparkbyexamples.com/apache-spark-rdd/spark-rdd-transformations/> (“**Spark RDD Transformations**”)
- [20] Shrey Mehrotra & Akash Grade, “Apache Quick Start Guide” (Packt Publishing, Jan. 2019), *available at* <https://learning.oreilly.com/library/view/apache-spark-quick/9781789349108/0ee1a5e2-09d0-49f0-99f5-9dee8336258d.xhtml> (“**Apache Spark Quick Start Guide**”)
- [21] “Spark SQL Shuffle Partitions,” *available at* <https://sparkbyexamples.com/spark/spark-shuffle-partitions/> (“**Spark SQL Shuffle Partitions**”)
- [22] “Apache Spark Map vs. FlatMap Operation,” DataFlair, *available at* <https://data-flair.training/blogs/apache-spark-map-vs-flatmap/> (“**Apache Spark Map vs. FlatMap**”)
- [23] Confluence Administrator, “LanguageManual Transform,” Dashboard (updated by Sushanth Sowmyan on Dec. 9, 2015), *available at* <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Transform> (“**LanguageManual Transform**”)

OVERVIEW



nelson bumgardner conroy

Representative Claim 1

- 1[pre]** A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:
- 1[a]** partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,
- 1[b]** wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and
- 1[c]** reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,
- 1[d]** wherein the mapping and reducing operations are performed by a distributed system.

CLAIM CHART



nelson bumgardner conroy

Claim Chart – Claim 1

1 [pre] A method of **processing data of a data set over a distributed system**, wherein the data set comprises a plurality of data groups, the method comprising:

Booking.com infringes Claim 1 of the '610 Patent. The Booking.com System performs a method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising the steps discussed herein. The Booking.com System is based on Hadoop + Hive + Spark, which enables processing of data of a data set over a distributed system. For example, Booking.com employs, e.g., Hadoop/Hive/Spark developers:

[14] Booking.com Big Data Engineer:



Big Data Engineer

Booking.com · Full-time

Nov 2019 – Jun 2021 · 1 yr 8 mos
Manchester, Reino Unido

Successfully implemented a process to move data from HDFS to AWS S3 in Spark (Scala)
Maintenance and enhancing workflows and processes using Oozie, Sqoop, Impala and Spark
Spark Streaming improvements together with Kafka

[5] Booking.com Lead Data Engineer

Since Booking.com started facilitating bookings in 1996, the amount of data produced and consumed has increased in unimaginable proportions (20TB/day), certainly from the perspective of our founders. The last decade, open source data tools (Hive, Spark, Cassandra and Kafka) running on large internal server parks enabled hundreds of colleagues working closely with data to produce various data products, e.g. in Machine Learning and Analytics. As the community has grown, so have the number of challenges around working with data. Providing flexible compute resources introduced the onset of clouds in parallel to a heavily utilized on-premise environment. Governments introduce standards for personal data protection. A growing, physically disconnected employee-base is less able to share tribal knowledge regarding data finesses. Hence, the establishment of 'Enterprise Data Management', a group governing the production and consumption of data, for it to be trusted and understood.

The Lead Data Engineer is a technical leader who drives broad data engineering strategies and delivery across a business area. You will lead solution envisaging, technical designs, hands-on implementation as well as provide operational support across multiple data domains. You need to influence, differentiate, and guide the business and technology strategies in your area, as they relate to data, through constant interaction with various teams. You ask the right questions to the right people in order to align data strategy with commercial strategy, demonstrating deep technical expertise and broad business knowledge. You play an active role in identifying data engineering skill gaps within your area and support development of tools, materials, and training to bridge these gaps.

Apply Now

Location
Amsterdam, The Netherlands


Skills

- Kafka
- Spark
- Scala
- Hadoop
- Python
- Java

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system , wherein the data set comprises a plurality of data groups, the method comprising:	<p>In another example, Booking.com employs, e.g., developers to implement Hadoop framework:</p> <p>[15] Booking.com Data Application Engineer:</p> <ul style="list-style-type: none">• 2+ years experience in management of bigger organisations (managing other managers)• Experience with Big Data Technologies (<u>Hadoop, Hive, Spark</u>, Oozie, GCP, etc.), data warehouses with Oracle, MySQL, PostgreSQL, etc• Demonstrated strength in SQL, data modelling, ETL development, and data warehousing• Experience with large-scale data warehousing and analytics projects• Experience mentoring and managing engineers on content, ensuring data engineering best practices are being followed• Independent, pragmatic and pro-active work attitude <p>As a Manager Software Development, you are responsible for leading a group of teams who develop and manage our Big Data Infrastructure. The teams are running <u>Hadoop clusters</u> of couple of thousands machines and petabytes of data, making sure we deliver a first class platform to our internal users.</p>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:</p>	<p>The Booking.com System is based on Hadoop + Hive + Spark, which enables processing of data of a data set over a distributed system.</p> <p>[10] Speed Up Your Queries with Hive LLAP Engine at 12:</p> <div data-bbox="772 544 2257 1332"> <p>Introducing Hive LLAP</p> <ul style="list-style-type: none"> • “Live Long and Process” or “Low Latency Analytical Processing” <ul style="list-style-type: none"> • Not an execution engine (like Tez), LLAP simply enhances the Hive execution model • Built for fast query response time against smaller data volumes • Allows concurrent execution of analytical workloads • Intelligent memory caching for quick startup and data sharing <ul style="list-style-type: none"> • Caches most active data in RAM • Shared cache across clients • Persistent server used to instantly execute queries <ul style="list-style-type: none"> • LLAP daemons are “always on” • Data passed to execution as it becomes ready  </div>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [pre] A method of processing data of a data set over a distributed system , wherein the data set comprises a plurality of data groups, the method comprising:	<p>The Booking.com System is implemented over a distributed system:</p> <p>[1] An introduction to Apache Hadoop for big data at 2:</p> <div><p>The Apache Hadoop framework is composed of the following modules</p><ol style="list-style-type: none">1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules2. <u>Hadoop Distributed File System (HDFS): a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster</u>3. <u>Hadoop YARN: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications</u>4. <u>Hadoop MapReduce: a programming model for large scale data processing</u><p><u>All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.</u> Apache Hadoop's MapReduce</p></div>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:</p>	<p>The Booking.com System is implemented over a distributed system:</p> <p>[11] Apache Hive:</p> <p><u>Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis.</u>^[3] Hive gives an <u>SQL-like interface</u> to query data stored in various databases and file systems that integrate with Hadoop. Traditional SQL queries must be implemented in the <u>MapReduce</u> Java API to execute SQL applications and queries over distributed data. Hive provides the necessary SQL abstraction to integrate SQL-like queries (HiveQL) into the underlying Java without the need to implement queries in the low-level Java API. Since most data warehousing applications work with SQL-based querying languages, Hive aids portability of SQL-based applications to Hadoop.^[4] While initially developed by <u>Facebook</u>, Apache Hive is used and developed by other companies such as <u>Netflix</u> and the <u>Financial Industry Regulatory Authority (FINRA)</u>.^{[5][6]} Amazon maintains a software fork of Apache Hive included in <u>Amazon Elastic MapReduce</u> on <u>Amazon Web Services</u>.^[7]</p>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:</p>	<p>Booking.com collects a myriad of data groups from a variety of sources (e.g., websites, apps, transactions, third parties, etc.):</p> <p>[16] Booking.com Privacy Statement:</p> <p>Booking.com collects and uses the info you provide us. When you make a Trip Reservation, you are (at a minimum) asked for <u>your name and email address</u>.</p> <p>Depending on the Trip Reservation, we may also ask for <u>your home address, telephone number, payment information, date of birth, current location (in the case of on-demand services), the names of the people traveling with you, and any preferences you might have for your Trip (such as dietary or accessibility requirements)</u>. In some cases, you may also be able to check-in online with the Trip Provider, for which we will ask you to share <u>passport information or a driver's license and signatures</u>.</p> <p>If you need to get in touch with our customer service team, contact your Trip Provider through us, or reach out to us in a different way (such as social media or via a chatbot); we'll collect information from you there, too. This applies whether you are contacting us with feedback or asking for help using our services.</p> <p>You might also be invited to write reviews to help inform others about the experiences you had on your Trip. When you write a review on the Booking.com platform, <u>we'll collect any info you've included, along with your display name and avatar (if you choose one)</u>.</p> <p>There are other instances where you'll provide us with information as well. For example, if you're browsing with your mobile device, you can decide to allow Booking.com to see <u>your current location or grant us access to some contact details</u>. This helps us to give you the best possible service and experience by, for example, showing you our city guides, suggesting the nearest restaurants or attractions to your location, or making other recommendations.</p> <p>If you create a user account, we'll also store your <u>personal settings, uploaded photos, and reviews of previous bookings</u>. This saved data can be used to help you plan and manage future Trip Reservations or benefit from other features only available to account holders, such as incentives or other benefits.</p> <p>We may offer you referral programs or sweepstakes. Participating in these will involve providing us with <u>relevant personal data</u>.</p>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [pre] A method of processing data of a data set over a distributed system, wherein the data set comprises a plurality of data groups, the method comprising:</p>	<p>[13] Apache Spark FAQ at 1:</p> <div data-bbox="769 372 2428 575"> <p>How does Spark relate to Apache Hadoop?</p> <p>Spark is a fast and general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, Hive, and any Hadoop InputFormat. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning.</p> </div> <p>[17] RDD Programming Guide at 4, 2:</p> <div data-bbox="848 691 2356 1208"> <p>Resilient Distributed Datasets (RDDs)</p> <p>Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: <u>parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</u></p> <p>Overview</p> <p>At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. <u>The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.</u> RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.</p> </div>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>The method performed by the Booking.com System includes partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group.</p> <p>For example, in the Booking.com System's Hadoop MapReduce framework, the input data is partitioned into independent chunks (data partitions), which are processed by Mapper functions:</p> <p>[2] MapReduce Tutorial at 2:</p> <div data-bbox="991 839 2023 1178" style="border: 1px solid black; padding: 10px;"> <p><u>A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner.</u> The framework sorts the outputs of the maps, which are then input to the <i>reduce tasks</i>. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.</p> </div>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>The data partitions with key-value pairs are passed to user-configurable Mapper functions for further processing:</p> <p>[4] HadoopMapReduce at 1:</p> <div data-bbox="838 539 2323 989" style="border: 1px solid black; padding: 10px;"> <p><u>As key-value pairs are read from the RecordReader they are passed to the configured Mapper. The user supplied Mapper does whatever it wants with the input pair and calls OutputCollector.collect with key-value pairs of its own choosing.</u> The output it generates must use one key class and one value class. This is because the Map output will be written into a SequenceFile which has per-file type information and all the records must have the same type (use subclassing if you want to output different data structures). The Map input and output key-value pairs are not necessarily related typewise or in cardinality.</p> </div>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>The Mapper function maps the key-value pairs to form intermediate key-value pairs:</p> <p>[2] MapReduce Tutorial at 7:</p> <div><p>Mapper</p><p>Mapper maps <u>input key/value pairs</u> to a set of <u>intermediate key/value pairs</u>.</p><p><u>Maps are the individual tasks that transform input records into intermediate records.</u> The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.</p></div>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>With Spark, a data set is first partitioned into a plurality of elements and distributed across nodes. This distribution of elements is called a resilient distributed dataset (RDD):</p> <p>[17] RDD Programming Guide at 2, 4, 5:</p> <div data-bbox="851 526 2216 1189"> <p>Overview</p> <p>At a high level, every Spark application consists of a <i>driver program</i> that runs the user's main function and executes various <i>parallel operations</i> on a cluster. The main abstraction Spark provides is a <i>resilient distributed dataset</i> (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to <i>persist</i> an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.</p> <p>Resilient Distributed Datasets (RDDs)</p> <p>Spark revolves around the concept of a <i>resilient distributed dataset</i> (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: <i>parallelizing</i> an existing collection in your driver program, or <i>referencing a dataset in an external storage system</i>, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.</p> <p>One important parameter for parallel collections is <i>the number of partitions to cut the dataset into</i>. Spark will run one task for each partition of the cluster. Typically you want 2-4 partitions for each CPU in your cluster. Normally, Spark tries to set the number of partitions automatically based on your cluster. However, you can also set it manually by passing it as a second parameter to <code>parallelize</code> (e.g. <code>sc.parallelize(data, 10)</code>). Note: some places in the code use the term <i>slices</i> (a synonym for partitions) to maintain backward compatibility.</p> </div>

Claim Chart – Claim 1

Claim Feature

1 [a] **partitioning the data of each one of the data groups into a plurality of data partitions** that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,

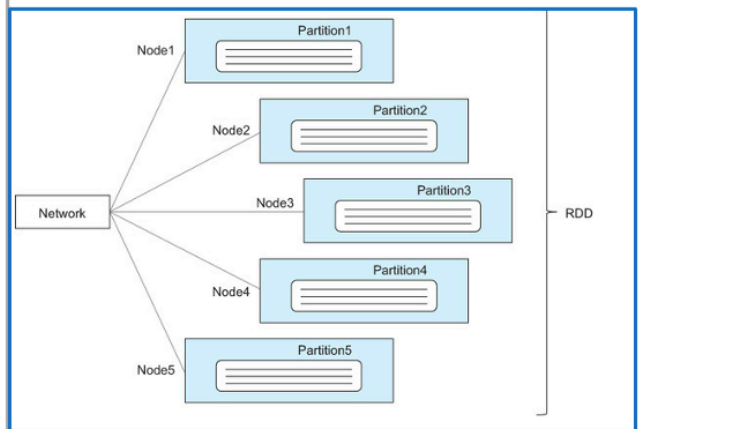
Evidence

[12]

Spark in Action:

Each part (piece or slice) of an RDD is called a *partition*.^[2] When you load a text file from your local filesystem into Spark, for example, the file's contents are split into partitions, which are evenly distributed to nodes in a cluster. More than one partition may end up on the same node. The sum of all those partitions forms your RDD. This is where the word *distributed* in *resilient distributed dataset* comes from. Figure 4.1 shows the distribution of lines of a text file loaded into an RDD in a five-node cluster. The original file had 15 lines of text, so each RDD partition was formed with 3 lines of text. Each RDD maintains a list of its partitions and an optional list of preferred locations for computing the partitions.

Figure 4.1. Simplified look at partitions of an RDD in a five-node cluster. The RDD was created by loading a text file using the `textFile` method of `SparkContext`. The loaded text file had 15 lines of text, so each partition was formed with 3 lines of text.



Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	<p>One form of the RDDs is a pair RDD, which includes key/value pairs:</p> <p>[18] Learning Spark:</p> <div>Spark provides special operations on <u>RDDs containing key/value pairs</u>. These RDDs are called <u>pair RDDs</u>. Pair RDDs are a useful building block in many programs, as they expose operations that allow you to act on each key in parallel or regroup data across the network. For example, pair RDDs have a <code>reduceByKey()</code> method that can aggregate data separately for each key, and a <code>join()</code> method that can merge two RDDs together by grouping elements with the same key. It is common to extract fields from an RDD (representing, for instance, an event time, customer ID, or other identifier) and use those fields as keys in pair RDD operations.</div> <p>[12] Spark in Action:</p> <div><p>4.1. WORKING WITH PAIR RDDS</p><p>Storing data as key-value pairs offers a simple, general, and extensible data model because each <u>key-value pair can be stored independently</u> and it's easy to add new types of keys and new types of values. This extensibility and simplicity have made this practice fundamental to several frameworks and applications. For example, many popular caching systems and NoSQL databases, such as memcached and Redis, are key-value stores. Hadoop's MapReduce also operates on key-value pairs (as you can see in appendix A).</p><p>Keys and values can be simple types such as integers and strings, as well as more-complex data structures. Data structures traditionally used to represent key-value pairs are <i>associative arrays</i>, also called <i>dictionaries</i> in Python and <i>maps</i> in Scala and Java.</p><p><u>In Spark, RDDs containing key-value tuples are called <i>pair RDDs</i></u>. Although you don't have to use data in Spark in the form of key-value pairs (as you've seen in the previous chapters), pair RDDs are well suited (and indispensable) for many use cases. <u>Having keys along with the data enables you to aggregate, sort, and join the data</u>, as you'll soon see. But before doing any of that, the first step is to create a pair RDD, of course.</p></div>

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>The RDDs support two types of data operations: transformations and actions.</p> <p>[17] RDD Programming Guide at 6:</p> <div data-bbox="792 501 2423 925"> <h3>RDD Operations</h3> <p>RDDs support two types of operations: <u>transformations</u>, which create a new dataset from an existing one, and <i>actions</i>, which return a value to the driver program after running a computation on the dataset. <u>For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results.</u> On the other hand, <i>reduce</i> is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel <i>reduceByKey</i> that returns a distributed dataset).</p> <p>All transformations in Spark are <i>lazy</i>, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, we can realize that a dataset created through <i>map</i> will be used in a <i>reduce</i> and return only the result of the <i>reduce</i> to the driver, rather than the larger mapped dataset.</p> </div>

Claim Chart – Claim 1

Claim Feature	Evidence																		
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function’s corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	<p>Examples of transformation functions:</p> <p>[17] RDD Programming Guide at 11–12:</p> <div><h3>Transformations ↗</h3><p>The following table lists some of the common transformations supported by Spark. Refer to the RDD API doc (Scala, Java, Python, R) and pair RDD functions doc (Scala, Java) for details.</p><table><tr><th>Transformation</th><th>Meaning</th></tr><tr><td><code>map(func)</code></td><td>Return a new distributed dataset formed by passing each element of the source through a function <i>func</i>.</td></tr><tr><td><code>filter(func)</code></td><td>Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.</td></tr><tr><td><code>flatMap(func)</code></td><td>Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).</td></tr><tr><td><code>mapPartitions(func)</code></td><td>Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.</td></tr><tr><td><code>mapPartitionsWithIndex(func)</code></td><td>Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.</td></tr></table><table><tr><td><code>groupByKey([numPartitions])</code></td><td>When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.</td></tr><tr><td><code>reduceByKey(func, [numPartitions])</code></td><td>When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i>, which must be of type <code>(V,V) => V</code>. Like in <code>groupByKey</code>, the number of reduce tasks is configurable through an optional second argument.</td></tr><tr><td><code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code></td><td>When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral “zero” value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code>, the number of reduce tasks is configurable through an optional second argument.</td></tr></table></div>	Transformation	Meaning	<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .	<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.	<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).	<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.	<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.	<code>groupByKey([numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.	<code>reduceByKey(func, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.	<code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral “zero” value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.
Transformation	Meaning																		
<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .																		
<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.																		
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).																		
<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.																		
<code>mapPartitionsWithIndex(func)</code>	Similar to mapPartitions, but also provides <i>func</i> with an integer value representing the index of the partition, so <i>func</i> must be of type <code>(Int, Iterator<T>) => Iterator<U></code> when running on an RDD of type T.																		
<code>groupByKey([numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.																		
<code>reduceByKey(func, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <i>func</i> , which must be of type <code>(V,V) => V</code> . Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.																		
<code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral “zero” value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.																		

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>The transformation operations include narrow and wide transformations. Wide transformations (also called "shuffle") operate on data from multiple partitions. Wide transformations include aggregateByKey, reduceByKey, etc.</p> <p>[19] Spark RDD Transformations at 2–3:</p> <div data-bbox="899 525 2175 1190"> <div data-bbox="937 572 1350 604"> <h3>RDD Transformation Types</h3> </div> <div data-bbox="937 646 1350 671"> <p>There are two types are transformations.</p> </div> <div data-bbox="937 753 1263 785"> <h3>Narrow Transformation</h3> </div> <div data-bbox="937 825 1442 979"> <p>Narrow transformations are the result of <u>map()</u> and <u>filter()</u> functions and these compute data that live on a single partition meaning there will not be any data movement between partitions to execute narrow transformations.</p> </div> <div data-bbox="1633 568 2068 651"> <p>Functions such as <u>map()</u>, <u>mapPartition()</u>, <u>flatMap()</u>, <u>filter()</u>, <u>union()</u> are some examples of narrow transformation</p> </div> <div data-bbox="1633 689 1921 721"> <h3>Wider Transformation</h3> </div> <div data-bbox="1633 756 2107 959"> <p>Wider transformations are the result of <u>groupByKey()</u> and <u>reduceByKey()</u> functions and these compute data that live on many partitions meaning there will be data movements between partitions to execute wider transformations. Since these shuffles the data, they also called shuffle transformations.</p> </div> <div data-bbox="1625 1012 2107 1129"> <p>Functions such as <u>groupByKey()</u>, <u>aggregateByKey()</u>, <u>aggregate()</u>, <u>join()</u>, <u>repartition()</u> are some examples of a wider transformations.</p> </div> </div>

Claim Chart – Claim 1

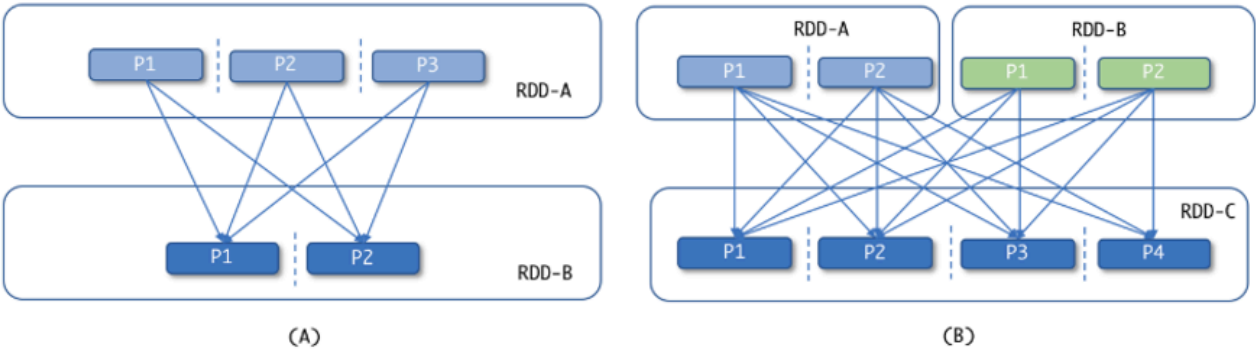
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and **providing each data partition to a selected one of a plurality of mapping functions** that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,

[20]

Apache Spark Quick Start Guide:

Wide transformations

Wide transformations involve a shuffle of the data between the partitions. The `groupByKey()`, `reduceByKey()`, `join()`, `distinct()`, and `intersect()` are some examples of wide transformations. In the case of these transformations, the result will be computed using data from multiple partitions and thus requires a shuffle. Wide transformations are similar to the shuffle-and-sort phase of MapReduce. Let's understand the concept with the help of the following example:



Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>When executing a shuffle transformation, a map task is first run on all data partitions.</p> <p>[21] Spark SQL Shuffle Partitions:</p> <div data-bbox="1503 378 2458 1206"> <h3>What is Spark Shuffle?</h3> <p>Shuffling is a mechanism Spark uses to <u>redistribute the data</u> across different executors and even across machines. Spark shuffling triggers when we perform certain transformation operations like <code>groupByKey()</code>, <code>reduceByKey()</code>, <code>join()</code> on RDD and DataFrame.</p> <p>When <u>creating an RDD</u> or DataFrame, Spark doesn't necessarily store the data for all keys in a partition since at the time of creation there is no way we can set the key for data set.</p> <p>Hence, when we run the <code>reduceByKey()</code> operation to aggregate the data on keys, Spark does the following. needs to first run tasks to collect all the data from all partitions and</p> <p>For example, when we perform <code>reduceByKey()</code> operation, Spark does the following</p> <ul style="list-style-type: none"> • Spark first runs <u>map tasks</u> on all partitions which groups all values for a single key. • The results of the map tasks are kept in memory. • When results do not fit in memory, Spark stores the data into a disk. • Spark shuffles the mapped data across partitions, some times it also stores the shuffled data into a disk for reuse when it needs to recalculate. • Run the garbage collection • Finally runs reduce tasks on each partition based on key. </div>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,	<p>[22] Apache Spark Map vs. FlatMap at 3–4:</p> <div><p>i. Spark Map Transformation</p><p>A map is a transformation operation in Apache Spark. It applies to each element of RDD and it returns the result as new RDD. <u>In the Map, operation developer can define his own custom business logic.</u> The same logic will be applied to all the elements of RDD.</p><p><u>Spark Map function takes one element as input process it according to custom code (specified by the developer) and returns one element at a time.</u> Map transforms an RDD of length N into another RDD of length N. The input and output RDDs will typically have the same number of records.</p></div>

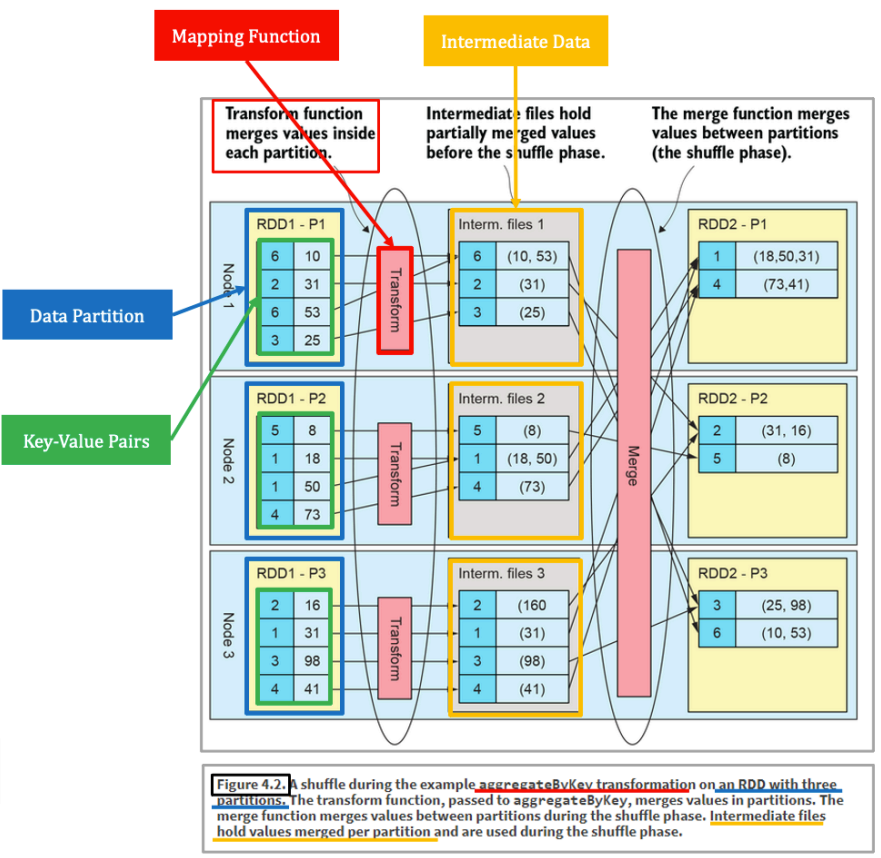
Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,</p>	<p>When a mapping function is applied, intermediate files are created for that partition.</p> <p>[12] Spark in Action:</p> <div data-bbox="1197 364 2446 785"> <p>4.2.2. Understanding and avoiding unnecessary shuffling</p> <p>Physical movement of data between partitions is called <i>shuffling</i>. It occurs when data from multiple partitions needs to be combined in order to build partitions for a new RDD. When grouping elements by key, for example, <u>Spark needs to examine all of the RDD's partitions, find elements with the same key, and then physically group them, thus forming new partitions.</u></p> <p>To visualize what happens with partitions during a shuffle, we'll use the previous example with the <u>aggregateByKey transformation</u> (from section 4.1.2). The shuffle that occurs during that transformation is illustrated in figure 4.2.</p> </div> <div data-bbox="1197 799 2446 1213"> <p><u>The transform function puts all values of each key in a single partition (partitions P1 to P3) into a list. Spark then writes this data to intermediate files on each node.</u> In the next phase, the merge function is called to merge lists from different partitions, but of the same key, into a single list for each key. The default partitioner (HashPartitioner) then kicks in and puts each key in its proper partition.</p> <p><u>Tasks that immediately precede and follow the shuffle are called <i>map</i> and <i>reduce</i> tasks, respectively.</u> The results of <u>map tasks are written to intermediate files</u> (often to the OS's filesystem cache only) and read by reduce tasks. In addition to being written to disk, the data is sent over the network, so it's important to try to minimize the number of shuffles during Spark jobs.</p> </div>

Claim Chart – Claim 1

1 [a] partitioning the data of each one of the data groups into a plurality of data partitions that each have a plurality of key-value pairs and providing each data partition to a selected one of a plurality of mapping functions that are each user-configurable to independently output a plurality of lists of values for each of a set of keys found in such map function's corresponding data partition to form corresponding intermediate data for that data group and identifiable to that data group,

In this example, the aggregateByKey transformation operation (shuffle) is performed on three exemplary partitions:
[12] Spark in Action:



Claim Chart – Claim 1

1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and

In the Booking.com System, the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common. For example, the Schema class in Hive is defined for each data set. Thus, the data of a first data group can have a different schema than the data of a second data group.

[7]

Modifier and Type

Class and Description

static class

Schema.Fields

The set of fields this struct contains, along with convenience methods for finding and manipulating them.

Claim Chart – Claim 1



1 [b] wherein the data of a first data group has a different schema than the data of a second data group and **the data of the first data group is mapped differently than the data of the second data group** so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and

Data partitions can be processed by using different user-configured Mappers:

[8] Hive Tutorial at 15:

Users can also plug in their own custom mappers and reducers in the data stream by using features natively supported in the Hive language, for example, in order to run a custom mapper script - map_script - and a custom reducer script - reduce_script - the user can issue the following command which uses the TRANSFORM clause to embed the mapper and the reducer scripts.

[4] HadoopMapReduce:

As key-value pairs are read from the RecordReader they are passed to the configured  Mapper. The user supplied Mapper does whatever it wants with the input pair and calls  OutputCollector.collect with key-value pairs of its own choosing. The output it generates must use one key class and one value class. This is because the Map output will be written into a SequenceFile which has per-file type information and all the records must have the same type (use subclassing if you want to output different data structures). The Map input and output key-value pairs are not necessarily related typewise or in cardinality.

Case 4:21-cv-00942 Document 1-6 Filed 11/29/21 Page 33 of 46 PageID #: 107

Claim Chart – Claim 1

Claim Feature	Evidence
1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and	<p>The Mapper function maps the key-value pairs to form intermediate key-value pairs. Similarly, different Mappers generate different intermediate results:</p> <p>[2] MapReduce Tutorial:</p> <div><p>Mapper maps input key/value pairs to a set of intermediate key/value pairs.</p><p><u>Maps are the individual tasks that transform input records into intermediate records.</u> The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.</p></div>

Claim Chart – Claim 1

1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data **have a key in common**; and

A Hive JOIN query may be used to join two datasets that have a key in common:

[9] **LanguageManual Joins:**

Examples

Some salient points to consider when writing join queries are as follows:

- Complex join expressions are allowed e.g.

```
SELECT a.* FROM a JOIN b ON (a.id = b.id)
```

```
SELECT a.* FROM a JOIN b ON (a.id = b.id AND a.department = b.department)
```

```
SELECT a.* FROM a LEFT OUTER JOIN b ON (a.id <> b.id)
```

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and</p>	<p>Spark supports multiple types of structure data, each with its own schema.</p> <div data-bbox="703 411 1156 446"> <p>[18] Learning Spark:</p> </div> <div data-bbox="1207 382 2288 679"> <p>Structured Data with Spark SQL</p> <p>Spark SQL is a component added in Spark 1.0 that is quickly becoming <u>Spark's preferred way to work with structured and semistructured data.</u> <u>By structured data, we mean data that has a <i>schema</i></u>—that is, a consistent set of fields across data records. <u>Spark SQL supports multiple structured data sources as input, and because it understands their schema,</u> it can efficiently read only the fields you require from these data sources. We will cover Spark SQL in more detail in <u>Chapter 9</u>, but for now, we show how to use it to load data from a few common sources.</p> </div> <div data-bbox="1207 686 2288 855"> <p>Apache Hive</p> <p><u>One common structured data source on Hadoop is Apache Hive. Hive can store tables in a variety of formats, from plain text to column-oriented formats, inside HDFS or other storage systems.</u> Spark SQL can load any table supported by Hive.</p> </div> <div data-bbox="1207 862 2288 1200"> <p>JSON</p> <p>If you have <u>JSON data with a consistent schema across records,</u> Spark SQL can infer their schema and load this data as rows as well, making it very simple to pull out the fields you need. To load JSON data, first create a HiveContext as when using Hive. (No installation of Hive is needed in this case, though—that is, you don't need a <i>hive-site.xml</i> file.) Then use the <code>HiveContext.jsonFile</code> method to get an RDD of Row objects for the whole file. Apart from using the whole Row object, you can also register this RDD as a table and select specific fields from it. For example, suppose that we had a JSON file containing tweets in the format shown in <u>Example 5-33</u>, one per line.</p> </div>

Claim Chart – Claim 1

1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, wherein the different schema and corresponding different intermediate data have a key in common; and

[18] Learning Spark:

File Formats

Spark makes it very simple to load and save data in a large number of file formats. Formats range from unstructured, like text, to semistructured, like JSON, to structured, like SequenceFiles (see Table 5-1). The input formats that Spark wraps all transparently handle compressed formats based on the file extension.

Table 5-1. Common supported file formats		
Format name	Structured	Comments
Text files	No	Plain old text files. Records are assumed to be one per line.
JSON	Semi	Common text-based format, semistructured; most libraries require one record per line.
CSV	Yes	Very common text-based format, often used with spreadsheet applications.
SequenceFiles	Yes	A common Hadoop file format used for key/value data.
Protocol buffers	Yes	A fast, space-efficient multilanguage format.
Object files	Yes	Useful for saving data from a Spark job to be consumed by shared code. Breaks if you change your classes, as it relies on Java Serialization.

Claim Chart – Claim 1

1 [b] wherein the data of a first data group has a different schema than the data of a second data group and **the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data**, wherein the different schema and corresponding different intermediate data have a key in common; and

[13] **Apache Spark FAQ at 1:**

How does Spark relate to Apache Hadoop?

Spark is a fast and general processing engine compatible with Hadoop data. It can run in Hadoop clusters through YARN or Spark's standalone mode, and it can process data in HDFS, HBase, Cassandra, **Hive**, and any Hadoop InputFormat. It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning.

[23] **LanguageManual Transform at 1:**

Transform/Map-Reduce Syntax

Users can also plug in their own custom mappers and reducers in the data stream by using features natively supported in the **Hive** language. e.g. in order to run a custom mapper script - map_script - and a custom reducer script - reduce_script - the user can issue the following command which uses the TRANSFORM clause to embed the mapper and the reducer scripts.

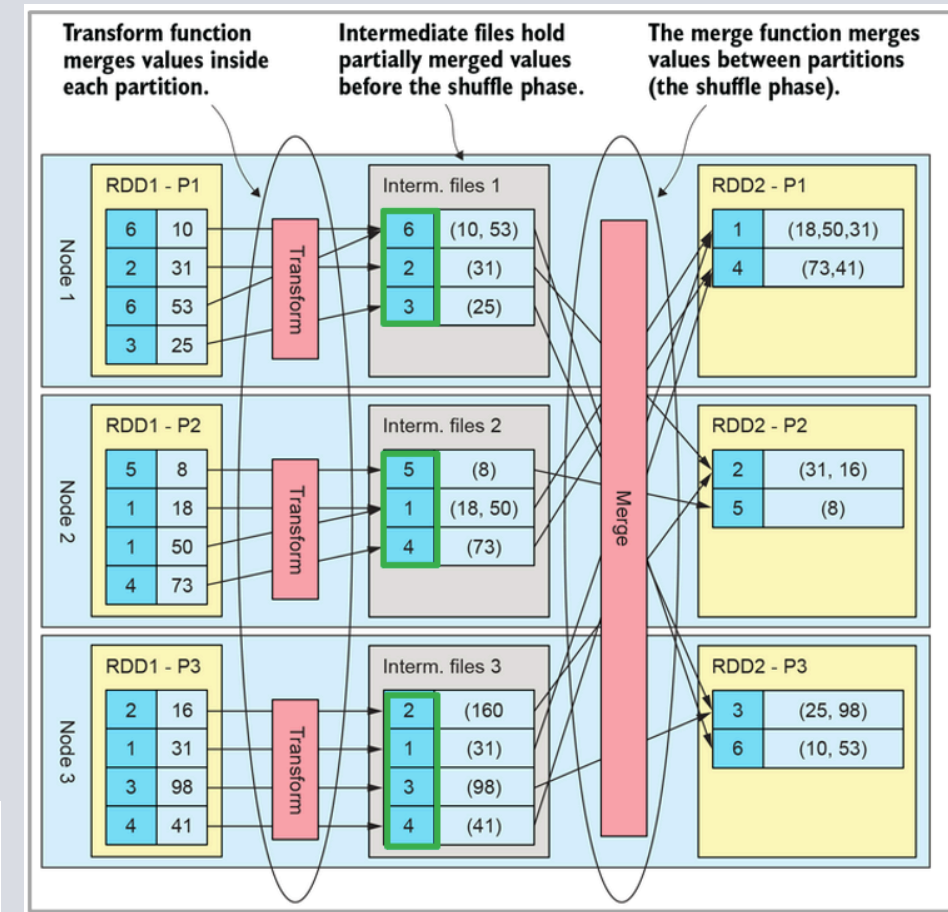
Claim Chart – Claim 1

1 [b] wherein the data of a first data group has a different schema than the data of a second data group and the data of the first data group is mapped differently than the data of the second data group so that different lists of values are output for the corresponding different intermediate data, **wherein the different schema and corresponding different intermediate data have a key in common**; and

[12] **Spark in Action:**

Figure 4.2. A shuffle during the example `aggregateByKey` transformation on an RDD with three partitions. The transform function, passed to `aggregateByKey`, merges values in partitions. The merge function merges values between partitions during the shuffle phase. Intermediate files hold values merged per partition and are used during the shuffle phase.

Evidence



Claim Chart – Claim 1

1 [c] **reducing the intermediate data for the data groups to at least one output data group**, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,

The method performed by the Booking.com System includes reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common.

The Reducer function reduces the intermediate values into a smaller set of values. This function operates in three phases: (1) Shuffle; (2) Sort; and (3) Reduce. The Shuffle and Sort phases operate on the output of various Mappers and group data before the final Reduce phase based on a common key.

[2] **MapReduce Tutorial:**

Reducer

Reducer reduces a set of intermediate values which share a key to a smaller set of values.

The number of reduces for the job is set by the user via `Job.setNumReduceTasks(int)`.

Overall, Reducer implementations are passed the Job for the job via the `Job.setReducerClass(Class)` method and can override it to initialize themselves. The framework then calls `reduce(WritableComparable, Iterable<Writable>, Context)` method for each <key, (list of values)> pair in the grouped inputs. Applications can then override the `cleanup(Context)` method to perform any required cleanup.

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,</p>	<p>[3] Interface Reducer:</p> <div data-bbox="968 349 2204 1225" style="border: 1px solid black; padding: 10px;"> <p>Reduces a set of intermediate values which share a key to a smaller set of values.</p> <p>The number of Reducers for the job is set by the user via <code>JobConf.setNumReduceTasks(int)</code>. Reducer implementations can access the <code>JobConf</code> for the job via the <code>JobConfigurable.configure(JobConf)</code> method and initialize themselves. Similarly they can use the <code>Closeable.close()</code> method for de-initialization.</p> <p>Reducer has 3 primary phases:</p> <div data-bbox="1006 528 2204 771" style="border: 1px solid blue; padding: 5px;"> <p>1. Shuffle</p> <p>Reducer is input the grouped output of a Mapper. In the phase the framework, for each Reducer, fetches the relevant partition of the output of all the Mappers, via HTTP.</p> <p>2. Sort</p> <p>The framework groups Reducer inputs by keys (since different Mappers may have output the same key) in this stage.</p> <p>The shuffle and sort phases occur simultaneously i.e. while outputs are being fetched they are merged.</p> </div> <p>Secondary Sort</p> <p>If equivalence rules for keys while grouping the intermediates are different from those for grouping keys before reduction, then one may specify a <code>Comparator</code> via <code>JobConf.setOutputValueGroupingComparator(Class)</code>. Since <code>JobConf.setOutputKeyComparatorClass(Class)</code> can be used to control how intermediate keys are grouped, these can be used in conjunction to simulate <i>secondary sort on values</i>.</p> <p>For example, say that you want to find duplicate web pages and tag them all with the url of the "best" known example. You would set up the job like:</p> <ul style="list-style-type: none"> • Map Input Key: url • Map Input Value: document • Map Output Key: document checksum, url pagerank • Map Output Value: url • Partitioner: by checksum • OutputKeyComparator: by checksum and then decreasing pagerank • OutputValueGroupingComparator: by checksum <p>3. Reduce</p> <p>In this phase the <code>reduce(Object, Iterator, OutputCollector, Reporter)</code> method is called for each <key, (list of values)> pair in the grouped inputs.</p> </div>

Claim Chart – Claim 1

1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a **merging of the corresponding different intermediate data based on the key in common,**

The Reduce function operates on different sets of input data (intermediate data) that have a key in common. From this processing, the Reduce function populates an output file.

[3] **Interface Reducer:**

reduce

```
void reduce(K2 key,
            Iterator<V2> values,
            OutputCollector<K3,V3> output,
            Reporter reporter)
    throws IOException
```

Reduces values for a given key.

The framework calls this method for each <key, (list of values)> pair in the grouped inputs. Output values must be of the same type as input values. Input keys must not be altered. The framework will **reuse** the key and value objects that are passed into the reduce, therefore the application should clone the objects they want to keep a copy of. In many cases, all values are combined into zero or one value.

Output pairs are collected with calls to `OutputCollector.collect(Object, Object)`.

Applications can use the `Reporter` provided to report progress or just indicate that they are alive. In scenarios where the application takes a significant amount of time to process individual key/value pairs, this is crucial since the framework might assume that the task has timed-out and kill that task. The other way of avoiding this is to set `mapreduce.task.timeout` to a high-enough value (or even zero for no time-outs).

Parameters:

key - the key.

values - the list of values to reduce.

output - to collect keys and combined values.

reporter - facility to report progress.

Throws:

IOException

Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,</p>	<p>In Spark, the intermediate data is reduced to an output data group.</p> <p>[21] Spark SQL Shuffle Partitions:</p> <div data-bbox="1434 386 2390 1218" style="border: 1px solid #ccc; padding: 10px;"> <p>What is Spark Shuffle?</p> <p>Shuffling is a mechanism Spark uses to redistribute the data across different executors and even across machines. Spark shuffling triggers when we perform certain transformation operations like <code>groupByKey()</code>, <code>reduceByKey()</code>, <code>join()</code> on RDD and DataFrame.</p> <p>When creating an RDD or DataFrame, Spark doesn't necessarily store the data for all keys in a partition since at the time of creation there is no way we can set the key for data set.</p> <p>Hence, when we run the <code>reduceByKey()</code> operation to aggregate the data on keys, Spark does the following. needs to first run tasks to collect all the data from all partitions and</p> <p>For example, when we perform <code>reduceByKey()</code> operation, Spark does the following</p> <ul style="list-style-type: none"> • Spark first runs <i>map</i> tasks on all partitions which groups all values for a single key. • The results of the map tasks are kept in memory. • When results do not fit in memory, Spark stores the data into a disk. • Spark shuffles the mapped data across partitions, some times it also stores the shuffled data into a disk for reuse when it needs to recalculate. • Run the garbage collection • Finally runs reduce tasks on each partition based on key. </div>

Claim Chart – Claim 1

Claim Feature	Evidence
1 [c] reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group, so as to result in a merging of the corresponding different intermediate data based on the key in common,	<p>[12] Spark in Action:</p> <div><p>4.2.2. Understanding and avoiding unnecessary shuffling</p><p>Physical movement of data between partitions is called <i>shuffling</i>. It occurs when data from multiple partitions needs to be combined in order to build partitions for a new RDD. When grouping elements by key, for example, Spark needs to examine all of the RDD's partitions, find elements with the same key, and then physically group them, thus forming new partitions.</p><p>To visualize what happens with partitions during a shuffle, we'll use the previous example with the <u>aggregateByKey transformation</u> (from section 4.1.2). The shuffle that occurs during that transformation is illustrated in <u>figure 4.2.</u></p></div> <div><p>The transform function puts all values of each key in a single partition (partitions P1 to P3) into a list. Spark then writes this data to intermediate files on each node. <u>In the next phase, the merge function is called to merge lists from different partitions, but of the same key, into a single list for each key.</u> The default partitioner (HashPartitioner) then kicks in and puts each key in its proper partition.</p><p><u>Tasks that immediately precede and follow the shuffle are called map and reduce tasks,</u> respectively. <u>The results of map tasks are written to intermediate files (often to the OS's filesystem cache only) and read by reduce tasks.</u> In addition to being written to disk, the data is sent over the network, so it's important to try to minimize the number of shuffles during Spark jobs.</p></div>

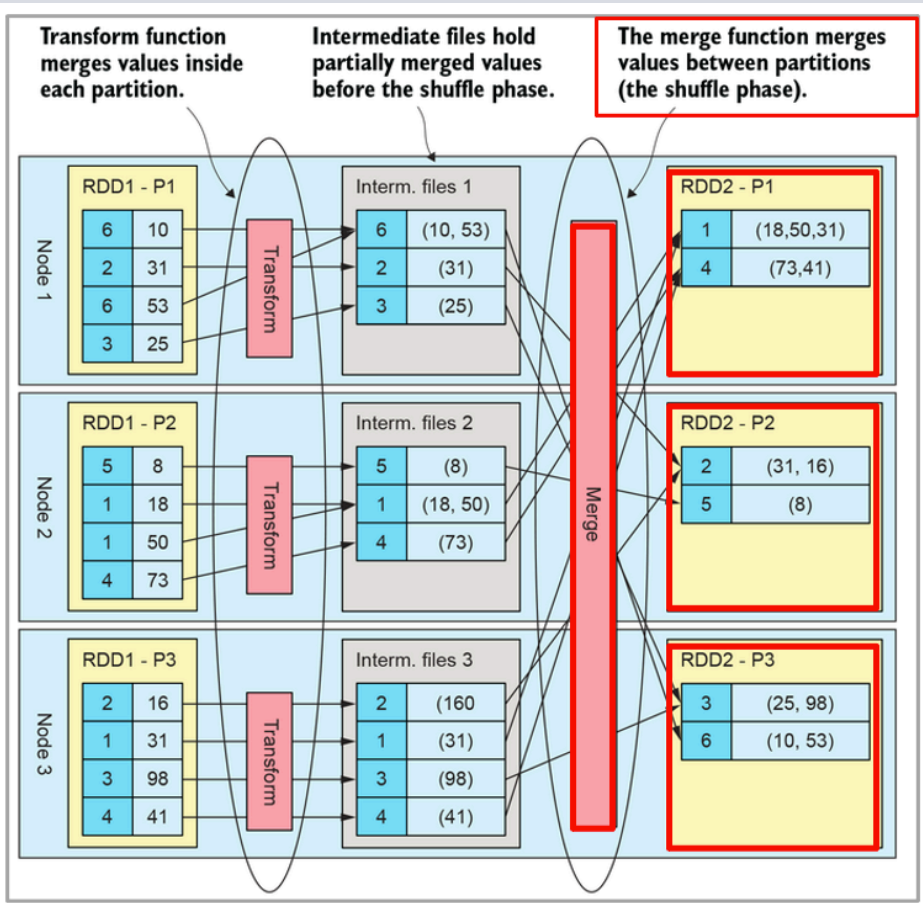
Claim Chart – Claim 1

1 [c] **reducing the intermediate data for the data groups to at least one output data group, including processing the intermediate data for each data group in a manner that is defined to correspond to that data group**, so as to result in a merging of the corresponding different intermediate data based on the key in common,

[12] **Spark in Action:**

Figure 4.2. A shuffle during the example `aggregateByKey` transformation on an RDD with three partitions. The transform function, passed to `aggregateByKey`, merges values in partitions. The merge function merges values between partitions during the shuffle phase. Intermediate files hold values merged per partition and are used during the shuffle phase.

Evidence



Claim Chart – Claim 1

Claim Feature	Evidence
<p>1 [d] wherein the mapping and reducing operations are performed by a distributed system.</p>	<p>The mapping and reducing operations are performed by the Booking.com System, which is a distributed system.</p> <p>[1] An introduction to Apache Hadoop for big data:</p> <div data-bbox="1274 508 2461 1179" style="border: 1px solid black; padding: 10px;"> <p>The Apache Hadoop framework is composed of the following modules</p> <ol style="list-style-type: none"> 1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules 2. <u>Hadoop Distributed File System (HDFS): a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster</u> 3. <u>Hadoop YARN: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications</u> 4. <u>Hadoop MapReduce: a programming model for large scale data processing</u> <p><u>All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.</u> Apache Hadoop's MapReduce</p> </div>

END



nelson bumgardner conroy